Augmenting Interpolation-Based Model Checking with Auxiliary Invariants

Dirk Beyer, **Po-Chun Chien**, and Nian-Ze Lee

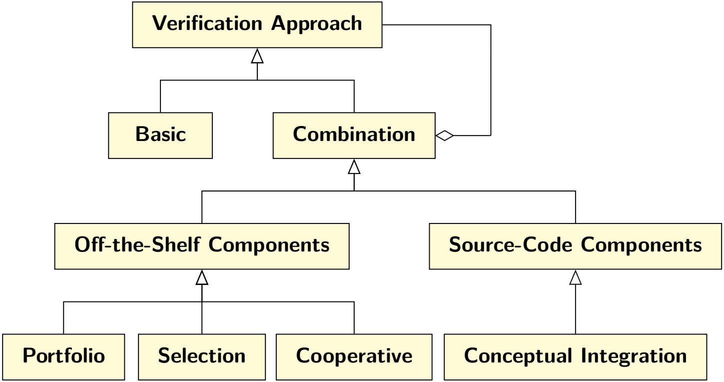
LMU Munich, Germany

COOP 2023-04-23

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

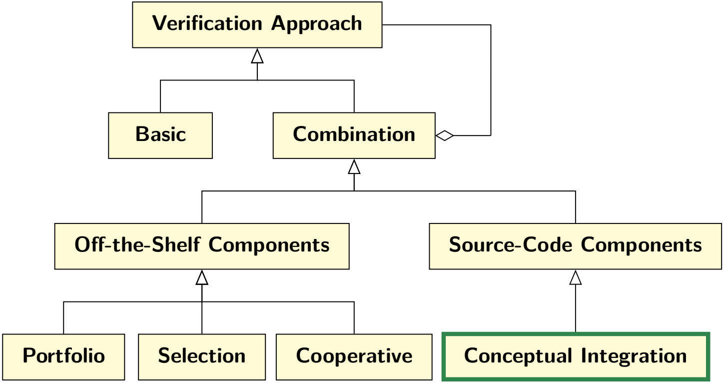
SoSy-Lab Software Systems

Cooperative Approaches



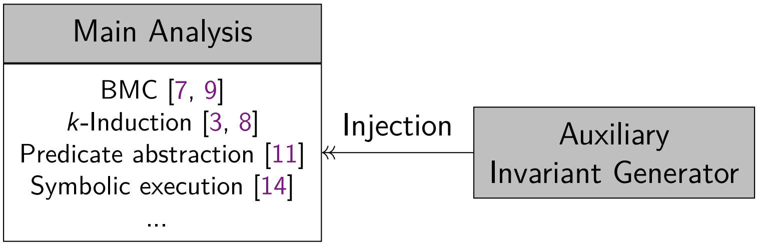
(Figure taken from Dirk Beyer's keynote talk at FTSCS 2022)

Cooperative Approaches

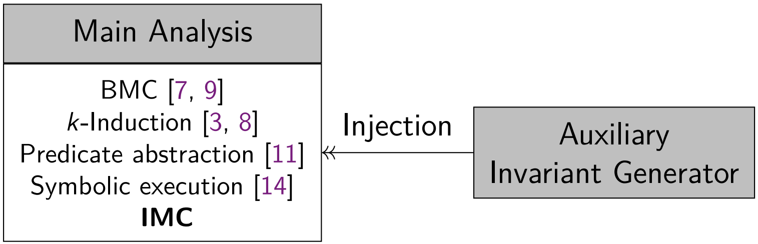


(Figure taken from Dirk Beyer's keynote talk at FTSCS 2022)

Cooperation via Invariant Injection



Cooperation via Invariant Injection



Interpolation and SAT-Based Model Checking

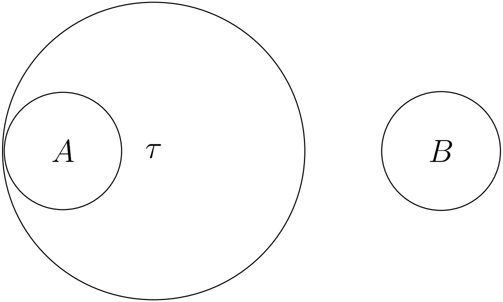
- K. L. McMillan, CAV 2003 [12]
- Interpolation-based model checking (IMC)
  - Originally designed for finite-state transition systems
  - Compute fixed points with interpolants derived from unsatisfiable BMC queries

Interpolation and SAT-Based Model Checking

- K. L. McMillan, CAV 2003 [12]
- Interpolation-based model checking (IMC)
  - Originally designed for finite-state transition systems
  - Compute fixed points with interpolants derived from unsatisfiable BMC queries
- State of the art for hardware verification
- Recently adopted for verifying software programs [6]

Craig Interpolation

- If $A(X, Y) \wedge B(Y, Z)$ is UNSAT: interpolant $\tau(Y)$
  - $A(X, Y) \Rightarrow \tau(Y)$
  - $\tau(Y) \wedge B(Y, Z)$ is UNSAT



Interpolation-Based Model Checking

- State-transition system: $I(s), T(s, s'), P(s)$

Interpolation-Based Model Checking

- State-transition system: $I(s), T(s, s'), P(s)$
- $I(s_0)T(s_0, s_1)\ T(s_1, s_2) \ldots T(s_{k-1}, s_k)\neg P(s_k)$

Interpolation-Based Model Checking

- State-transition system: $I(s), T(s, s'), P(s)$
- $\underbrace{I(s_0)T(s_0, s_1)}_{A_0(s_0,s_1)} \underbrace{T(s_1, s_2) \dots T(s_{k-1}, s_k) \neg P(s_k)}_{B(s_1,s_2,\dots,s_k)}$
- Interpolant $\tau_1(s_1)$: 1-step overapproximation

Interpolation-Based Model Checking

- State-transition system: $I(s), T(s, s'), P(s)$
- $\underbrace{I(s_0)T(s_0, s_1)}_{A_0(s_0,s_1)} \underbrace{T(s_1, s_2) \ldots T(s_{k-1}, s_k) \neg P(s_k)}_{B(s_1,s_2,\ldots,s_k)}$
- Interpolant $\tau_1(s_1)$: 1-step overapproximation
- $\underbrace{\tau_1(s_0)T(s_0, s_1)}_{A_1(s_0,s_1)} \underbrace{T(s_1, s_2) \ldots T(s_{k-1}, s_k) \neg P(s_k)}_{B(s_1,s_2,\ldots,s_k)}$
  - Interpolant $\tau_2(s_1)$: 2-step overapproximation
  - Repeat until $I \vee \bigvee \tau_i$ becomes a fixed point
  - Increment k if a query becomes satisfiable

Strengthen Interpolants with Auxililiary Invariants

- Given an **inductive** invariant Inv, interpolant τ_i can be strengthened by

$$\tau_i' = \tau_i \land Inv$$

Strengthen Interpolants with Auxililiary Invariants

- Given an **inductive** invariant Inv, interpolant τ_i can be strengthened by

$$\tau_i' = \tau_i \land Inv$$

- τ_i' is a valid interpolant for IMC

Strengthen Interpolants with Auxililiary Invariants

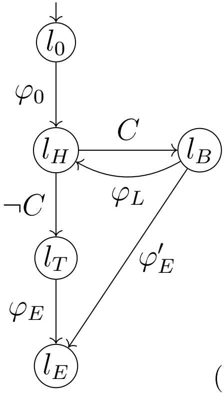
- Given an **inductive** invariant Inv, interpolant τ_i can be strengthened by

$$\tau_i' = \tau_i \wedge Inv$$

- τ_i' is a valid interpolant for IMC
- Inv helps remove some **unreachable** states in τ_i
- Adding more constraints $\rightarrow$ less likely to become SAT

IMC for Software Verification

- Summarize single-loop CFA by LBE [2, 6]



$$(l_0, \mathit{true})$$
$$\downarrow$$
$$(l_H, \varphi_0|_{i=0}) \;:\; I(s_0)$$
$$\downarrow$$
$$(l_H, (C \wedge \varphi_L)|_{i=1}) \;:\; T(s_0, s_1)$$
$$\downarrow$$
$$(l_H, (C \wedge \varphi_L)|_{i=2}) \;:\; T(s_1, s_2)$$
$$\vdots$$
$$(l_H, (C \wedge \varphi_L)|_{i=k}) \;:\; T(s_{k-1}, s_k)$$
$$\downarrow$$
$$(l_E, ((\neg C \wedge \varphi_E) \vee (C \wedge \varphi'_E))|_{i=k}) \;:\; \neg P(s_k)$$

(For multi-loop programs: standard transformation to single loop)

Collecting Formulas

```
int main(void) {
  unsigned x = 0;
  unsigned j = 0;
  while (nondet()) {
    x += 2;
    if (j == 3)
      x += 1;
    j += 1;
    if (j == 2)
      j = 0;
  }
  if (x % 2) {
    ERROR: return 1;
  }
  return 0;
}
```

- $s = \{x, j\}$

Collecting Formulas

I

```
int main(void) {
  unsigned x = 0;
  unsigned j = 0;
  while (nondet()) {
    x += 2;
    if (j == 3)
      x += 1;
    j += 1;
    if (j == 2)
      j = 0;
  }
  if (x % 2) {
    ERROR: return 1;
  }
  return 0;
}
```

- $s = \{x, j\}$
- $I(s)$: $(x = 0) \land (j = 0)$

Collecting Formulas

```c
int main(void) {
  unsigned x = 0;
  unsigned j = 0;
  while (nondet()) {
    x += 2;
    if (j == 3)
      x += 1;
    j += 1;
    if (j == 2)
      j = 0;
  }
  if (x % 2) {
    ERROR: return 1;
  }
  return 0;
}
```

- $s = \{x, j\}$
- $I(s)$: $(x = 0) \land (j = 0)$
- $T(s_i, s_{i+1})$:

$$
\begin{aligned}
&(x'_i = x_i + 2) \\
&\land(j_i = 3 \Rightarrow x_{i+1} = x'_i + 1) \\
&\land(j_i \neq 3 \Rightarrow x_{i+1} = x'_i) \\
&\land(j'_i = j_i + 1) \\
&\land(j'_i = 2 \Rightarrow j_{i+1} = 0) \\
&\land(j'_i \neq 2 \Rightarrow j_{i+1} = j'_i)
\end{aligned}
$$

Collecting Formulas

```
  int main(void) {
I   unsigned x = 0;
    unsigned j = 0;
    while (nondet()) {
      x += 2;
      if (j == 3)
T       x += 1;
      j += 1;
      if (j == 2)
        j = 0;
    }
    if (x % 2) {
P     ERROR: return 1;
    }
    return 0;
  }
```

- $s = \{x, j\}$
- $I(s)$: $(x = 0) \land (j = 0)$
- $T(s_i, s_{i+1})$:
  $$(x'_i = x_i + 2)$$
  $$\land(j_i = 3 \Rightarrow x_{i+1} = x'_i + 1)$$
  $$\land(j_i \neq 3 \Rightarrow x_{i+1} = x'_i)$$
  $$\land(j'_i = j_i + 1)$$
  $$\land(j'_i = 2 \Rightarrow j_{i+1} = 0)$$
  $$\land(j'_i \neq 2 \Rightarrow j_{i+1} = j'_i)$$
- $P(s)$: $x\%2 = 0$

Plain IMC Example Run

```c
int main(void) {
  unsigned x = 0;
  unsigned j = 0;
  while (nondet()) {
    x += 2;
    if (j == 3)
      x += 1;
    j += 1;
    if (j == 2)
      j = 0;
  }
  if (x % 2) {
    ERROR: return 1;
  }
  return 0;
}
```

One loop unrolling ($k = 1$)

- $I \wedge T \wedge \neg P$ is UNSAT
- τ_1: $x \% 2 = 0$

Plain IMC Example Run

```
   int main(void) {
I    unsigned x = 0;
     unsigned j = 0;
     while (nondet()) {
       x += 2;
       if (j == 3)
T        x += 1;
       j += 1;
       if (j == 2)
         j = 0;
     }
     if (x % 2) {
P      ERROR: return 1;
     }
     return 0;
   }
```

One loop unrolling ($k = 1$)

- $I \land T \land \neg P$ is UNSAT
- τ_1: $x\%2 = 0$
- $\tau_1 \land T \land \neg P$ is SAT
- Increase k

Augmented IMC Example Run

```c
int main(void) {
```

I

```c
  unsigned x = 0;
  unsigned j = 0;
```

T

```c
  while (nondet()) {
    x += 2;
    if (j == 3)
      x += 1;
    j += 1;
    if (j == 2)
      j = 0;
  }
```

P

```c
  if (x % 2) {
    ERROR: return 1;
  }
```

```c
  return 0;
}
```

One loop unrolling ($k = 1$)

- Inv: $0 \leq j \leq 1$

Augmented IMC Example Run

```
   int main(void) {
I    unsigned x = 0;
     unsigned j = 0;
     while (nondet()) {
       x += 2;
       if (j == 3)
T        x += 1;
       j += 1;
       if (j == 2)
         j = 0;
     }
     if (x % 2) {
P      ERROR: return 1;
     }
     return 0;
   }
```

One loop unrolling ($k = 1$)

- Inv: $0 \leq j \leq 1$
- $I \wedge T \wedge \neg P$ is UNSAT
- τ_1: $x\%2 = 0$
- τ_1': $\tau_1 \wedge Inv$
- $\tau_1' \wedge T \wedge \neg P$ is UNSAT

Augmented IMC Example Run

```
    int main(void) {
I     unsigned x = 0;
      unsigned j = 0;
T     while (nondet()) {
        x += 2;
        if (j == 3)
          x += 1;
        j += 1;
        if (j == 2)
          j = 0;
      }
P     if (x % 2) {
        ERROR: return 1;
      }
      return 0;
    }
```

One loop unrolling ($k = 1$)

- Inv: $0 \leq j \leq 1$
- $I \wedge T \wedge \neg P$ is UNSAT
- τ_1: $x\%2 = 0$
- τ_1': $\tau_1 \wedge Inv$
- $\tau_1' \wedge T \wedge \neg P$ is UNSAT
- τ_2: $x\%2 = 0$
- τ_2': $\tau_2 \wedge Inv$

Augmented IMC Example Run

```
    int main(void) {
I     unsigned x = 0;
      unsigned j = 0;
T     while (nondet()) {
        x += 2;
        if (j == 3)
          x += 1;
        j += 1;
        if (j == 2)
          j = 0;
      }
P     if (x % 2) {
        ERROR: return 1;
      }
      return 0;
    }
```

One loop unrolling ($k = 1$)

- Inv: $0 \leq j \leq 1$
- $I \wedge T \wedge \neg P$ is UNSAT
- τ_1: $x\%2 = 0$
- τ_1': $\tau_1 \wedge Inv$
- $\tau_1' \wedge T \wedge \neg P$ is UNSAT
- τ_2: $x\%2 = 0$
- τ_2': $\tau_2 \wedge Inv$
- $\tau_2' \equiv \tau_1'$: fixed point!

Evaluation

We conducted experiments to answer the following research questions

- **RQ1**: Can auxiliary invariants help improve IMC?

Evaluation

We conducted experiments to answer the following research questions

- **RQ1**: Can auxiliary invariants help improve IMC?
  - Reduce #unrollings and #interpolation-queries?
    Yes, there are cases with $>$ 10x improvement

Evaluation

We conducted experiments to answer the following research questions

- **RQ1**: Can auxiliary invariants help improve IMC?
  - Reduce #unrollings and #interpolation-queries?
    Yes, there are cases with >10x improvement
  - Boost run-time efficiency?
    Yes, especially the walltime for solving harder tasks

Evaluation

We conducted experiments to answer the following research questions

- **RQ1**: Can auxiliary invariants help improve IMC?
  - Reduce #unrollings and #interpolation-queries?
    Yes, there are cases with >10x improvement
  - Boost run-time efficiency?
    Yes, especially the walltime for solving harder tasks
- **RQ2**: Is the augmented IMC competitive?
  Yes, more proofs compared to other SMT-based algorithms

Invariant Generator

- Continuously-refining data-flow analysis (DF) based on intervals [3, 4]
- Invariants are expressions over intervals
  - e.g. $(0 \leq j \leq 1) \land (x < 5 \lor x > 7)$
- Invariant injection denoted as $\leftarrow\!\circlearrowleft\!-$DF

Tool Configurations and Benchmarks

- CPAchecker[1] revision 42901 of branch *imc-with-invariants*
- Interpolants computed by MathSAT5

[1] https://cpachecker.sosy-lab.org/

Tool Configurations and Benchmarks

- CPAchecker[1] revision 42901 of branch *imc-with-invariants*
- Interpolants computed by MathSAT5
- Compared SMT-based algorithms
  - IMC [6] vs. IMC←↺—DF
  - KI←↺—DF [3], predicate abstraction [10], Impact [13]

[1] https://cpachecker.sosy-lab.org/

Tool Configurations and Benchmarks

- CPAchecker[1] revision 42901 of branch *imc-with-invariants*
- Interpolants computed by MathSAT5
- Compared SMT-based algorithms
  - IMC [6] vs. IMC←↺−DF
  - KI←↺−DF [3], predicate abstraction [10], Impact [13]
- Safe tasks from *ReachSafety* of SV-COMP ’22 [1]
  - Eliminate easy tasks solved by BMC within 900 s
  - 1623 tasks remaining, 870 with non-trivial invariants

[1] https://cpachecker.sosy-lab.org/

Experimental Setup

- Environment
  - OS: Ubuntu 22.04 (64 bit)
  - Machine: 3.4 GHz CPU (8 cores) and 33 GB of RAM
- Each task is limited to
  - 4 CPU cores
  - 900 s of CPU time (max 150 s for DF)
  - 15 GB of RAM

  (reliable resource management by BenchExec[2])

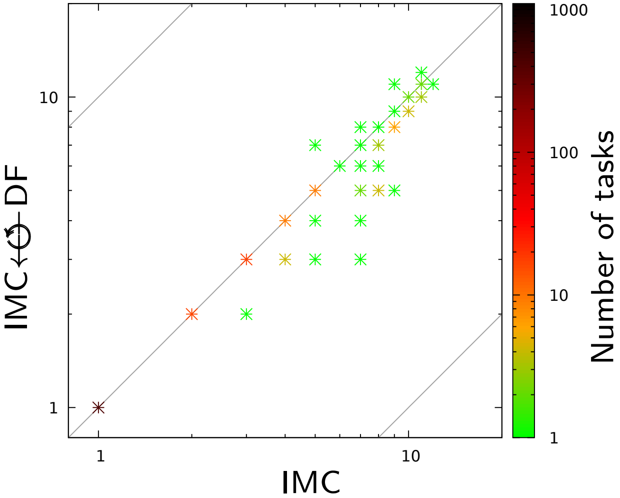
[2] https://github.com/sosy-lab/benchexec

Tasks with Significant Improvement

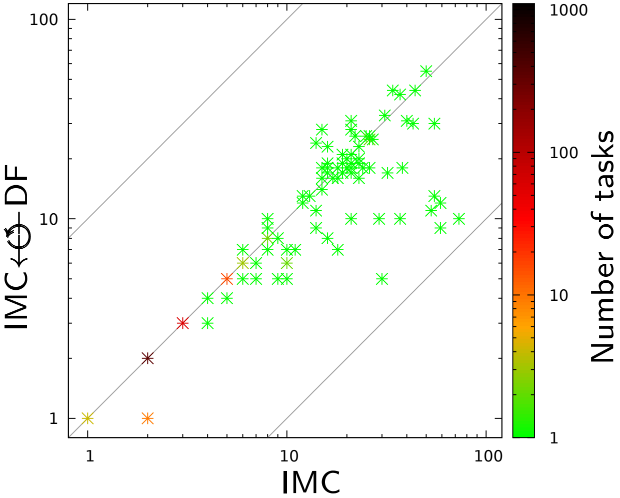
Task	IMC (timeout) #unroll	#itp	walltime	IMC↺DF (solved) #unroll	#itp	walltime
benchmark37_conj	318	316	892	2	2	1.83
s3_srvr_1a.BV.c.cil	65	441	885	6	13	6.26
Problem03_label51	11	57	878	6	11	24.3
Problem03_label15	8	54	876	6	11	35.4
Problem03_label03	9	50	872	6	11	27.2
s3_srvr_2a_alt.BV	38	278	883	20	125	71.7
Problem05_label12	13	42	876	10	32	400
s3_srvr_2a.BV.c.cil	40	290	887	36	252	635
Problem05_label50	13	39	878	11	37	531
Problem05_label07	12	50	873	12	48	699

(time unit: s)

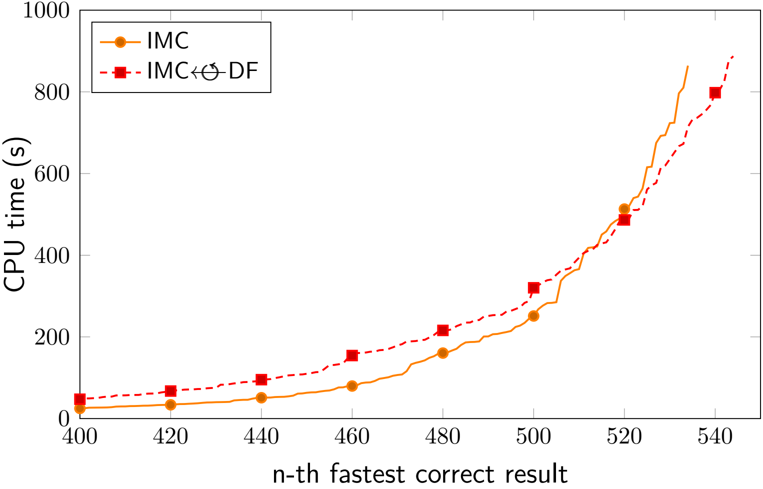
Scatter Plot: #Unrollings



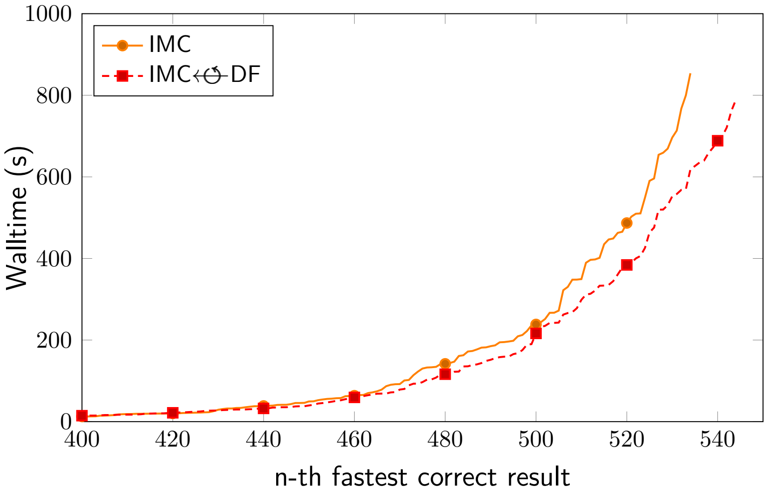
Scatter Plot: #Interpolation-Queries



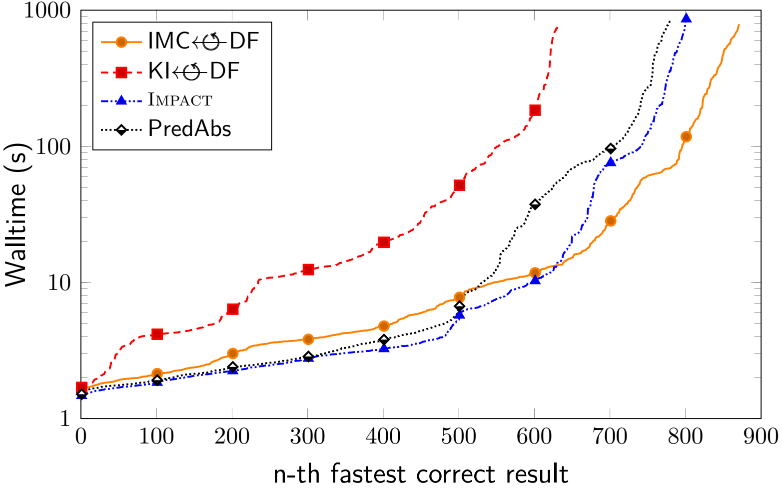
Quantile Plot: CPU time



Quantile Plot: Walltime



Quantile Plot: Comparsion with Others



Conclusion

- Augment IMC via invariant injection
- Open-source implementation in CPAchecker
- In our evaluation, the proposed method can
  - Reduce program unrollings and interpolation queries
  - Improve walltime efficiency
  - Find more proofs

References I

[1] Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). pp. 375–402. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_20

[2] Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351147

[3] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_42

[4] Beyer, D., Dangl, M., Wendler, P.: Combining k-induction with continuously-refined invariants. Tech. Rep. MIP-1503, University of Passau (January 2015), arXiv:1502.00096

[5] Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010), https://www.sosy-lab.org/research/pub/2010-FMCAD.Predicate_Abstraction_with_Adjustable-Block_Encoding.pdf

[6] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. arXiv/CoRR **2208**(05046) (July 2022). https://doi.org/10.48550/arXiv.2208.05046

References II

[7] Cheng, X., Hsiao, M.S.: Simulation-directed invariant mining for software verification. In: Proc. DATE. pp. 682–687. ACM (2008). https://doi.org/10.1109/DATE.2008.4484757

[8] Donaldson, A.F., Haller, L., Kröning, D.: Strengthening induction-based race checking with lightweight static analysis. In: Proc. VMCAI. pp. 169–183. LNCS 6538, Springer (2011). https://doi.org/10.1007/978-3-642-18275-4_13

[9] Ganai, M.K., Gupta, A.: Accelerating high-level bounded model checking. In: Proc. ICCAD. pp. 794–801. ACM (2006). https://doi.org/10.1145/1233501.1233664

[10] Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: Proc. POPL. pp. 232–244. ACM (2004). https://doi.org/10.1145/964001.964021

[11] Jain, H., Ivancic, F., Gupta, A., Shlyakhter, I., Wang, C.: Using statically computed invariants inside the predicate abstraction and refinement loop. In: Proc. CAV. pp. 137–151. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_15

[12] McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725, Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_1

References III

[13] McMillan, K.L.: Lazy abstraction with interpolants. In: Proc. CAV. pp. 123–136. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_14

[14] Pasareanu, C.S., Visser, W.: Verification of Java programs using symbolic execution and invariant generation. In: Proc. SPIN. pp. 164–181. LNCS 2989, Springer (2004). https://doi.org/10.1007/978-3-540-24732-6_13

Adopting IMC for Software Verification

- System under verification $\rightarrow I(s), T(s, s'), P(s)$
  - Sequential circuit: monolithic loop
  - Program: arbitrary control flow
- Solution: Use large-block encoding (LBE) [2, 5] to summarize control-flow automaton (CFA)
  - Loop-free blocks replaced by single transitions

IMC: Main Procedure

Input: $\mathbb{D} = \mathbb{L} \times \mathbb{P} \times \mathbb{LB}$ and k_{max}
Output: **false** if l_E reachable; **true** if fixed point obtained; **unknown** otherwise

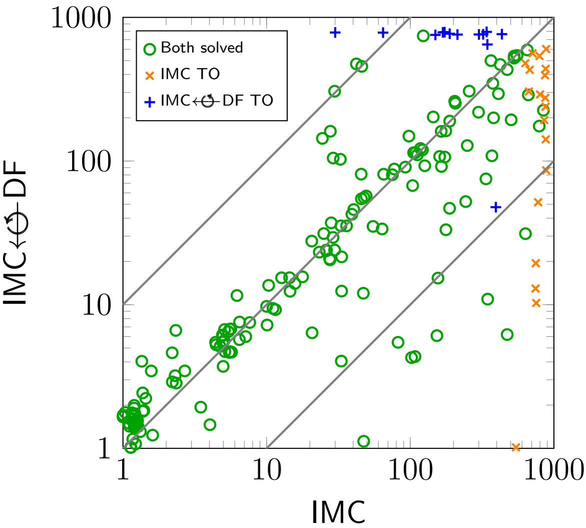
$k := 1$
$e_0 := (l_0, (true, l_0, true, true), \{l_H \mapsto -1\})$
reached := waitlist := $\{e_0\}$
while $k \leq k_{max}$ **do**
    (reached, waitlist) := CPA++($\mathbb{D}$, reached, waitlist, k)
    $(\sigma_p, \sigma_l, \sigma_s)$:= collect_formulas(reached, k)
    if sat($\sigma_p \wedge \sigma_l \wedge \sigma_s$) **then**
        return false
    if $k > 1$ **and** reach_fixed_point(σ_p,σ_l,σ_s) **then**
        return true
    $k := k + 1$
return unknown

IMC: reach_fixed_point(σ_p,σ_l,σ_s)

Input: σ_p, σ_l, and σ_s
Output: **true** if fixed point obtained; **false** otherwise

image := start := σ_p
while $\neg$sat(start $\wedge\ \sigma_l \wedge \sigma_s$) **do**
    τ := get_interpolant(start $\wedge\ \sigma_l, \sigma_s$)
    τ := shift_variable_index(τ, σ_p)
    if $\neg$sat($\tau \wedge \neg$image) **then**
        return true
    image := image $\vee\ \tau$
    start := τ
return false

Scatter Plot: Interpolation-Time



Scatter Plot: Wallime